

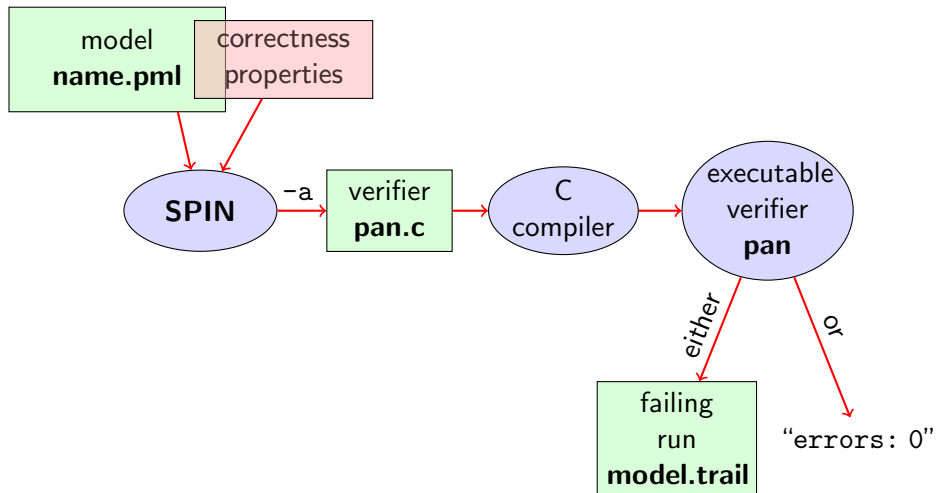
Software Engineering using Formal Methods

Model Checking with Temporal Logic

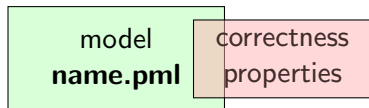
Wolfgang Ahrendt

23th September 2014

Model Checking with SPIN



Stating Correctness Properties



Correctness properties can be stated **within**, or **outside**, the model.

stating properties within model using

- ▶ assertion statements ✓
- ▶ meta labels
 - ▶ end labels ✓
 - ▶ **accept labels** (briefly)
 - ▶ progress labels

stating properties outside model using

- ▶ **never claims** (briefly)
- ▶ **temporal logic formulas** (today's main topic)

1. Accept labels in PROMELA \leftrightarrow Büchi automata
2. Fairness

Preliminaries 1: Acceptance Cycles

Definition (Accept Location)

A location marked with an **accept label** of the form “accept.xxx:” is called an **accept location**.

Accept locations can be used to **specify cyclic behavior**

Definition (Acceptance Cycle)

A run which **infinitely often** passes through an **accept location** is called an **acceptance cycle**.

Acceptance cycles are mainly used in **never claims** (see below), to define forbidden infinite behavior

Preliminaries 2: Fairness

Does this PROMELA model terminate in each run?

Demo: start/fair.pml

```
byte n = 0;
bool flag = false;

active proctype P() {
  do :: flag -> break
     :: else -> n = 5 - n
  od
}
active proctype Q() {
  flag = true
}
```

Termination guaranteed only if scheduling is (weakly) **fair**!

Definition (Weak Fairness)

A run is called **weakly fair** iff the following holds:
each **continuously executable** statement is **executed eventually**.

Model Checking of Temporal Properties

Many correctness properties not expressible by assertions

- ▶ all properties that involve state changes
- ▶ temporal logic expressive enough to characterize many (but not all) properties

In this course: “temporal logic” synonymous with “linear temporal logic”

Today: model checking of properties formulated in **temporal logic**

Beyond Assertions

Locality of Assertions

Assertions talk only about the state at their location in the code

Example

Mutual exclusion enforced by adding assertion to **each** critical section

```
critical++;  
assert( critical <= 1 );  
critical--;
```

Drawbacks

- ▶ no separation of concerns (model vs. correctness property)
- ▶ changing assertions is error prone (easily out of sync)
- ▶ easy to forget assertions:
correctness property might be violated at unexpected locations
- ▶ **many interesting properties not expressible via assertions**

Temporal Correctness Properties

Examples of properties more conveniently expressed as **global** properties than as assertions:

Mutual Exclusion

“critical ≤ 1 holds **throughout any run**”

Array Index within Bounds (given array a of length len)

“ $0 \leq i \leq len-1$ holds **throughout any run**”

Examples of properties **impossible** to express as assertions:

Absence of Deadlock

“If several processes try to enter their critical section, **eventually one of them** does so.”

Absence of Starvation

“If one process tries to enter its critical section, **eventually that process** does so.”

All of these are temporal properties \Rightarrow **use temporal logic**

Boolean Temporal Logic

Numerical variables in expressions

- ▶ Expressions such as $i \leq \text{len}-1$ contain numerical variables
- ▶ Propositional LTL as introduced so far only knows propositions
- ▶ Slight generalisation of LTL required

In **Boolean Temporal Logic** atomic building blocks are
Boolean expressions over PROMELA variables

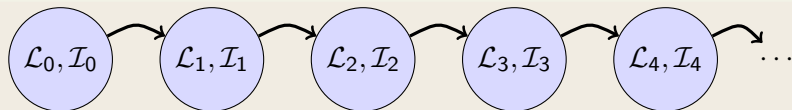
Boolean Temporal Logic over PROMELA

Set For_{BTL} of **Boolean Temporal Formulas** (simplified)

- ▶ all **global** PROMELA **variables** and **constants** of type **bool/bit** are $\in For_{BTL}$
- ▶ if e_1 and e_2 are numerical PROMELA expressions, then all of **$e_1==e_2$, $e_1!=e_2$, $e_1<e_2$, $e_1<=e_2$, $e_1>e_2$, $e_1>=e_2$** are $\in For_{BTL}$
- ▶ if P is a process and l is a label in P , then **$P@l$** is $\in For_{BTL}$ ($P@l$ reads "P is at l")
- ▶ if ϕ and ψ are formulas $\in For_{BTL}$, then all of
$$! \phi, \quad \phi \ \&\& \ \psi, \quad \phi \ || \ \psi, \quad \phi \ \rightarrow \ \psi, \quad \phi \ \leftrightarrow \ \psi$$
$$[\] \phi, \quad < > \phi, \quad \phi \ U \ \psi$$
are $\in For_{BTL}$

Semantics of Boolean Temporal Logic

A run σ through a PROMELA model M is a chain of states



- ▶ \mathcal{L}_j maps each running process to its current location counter
- ▶ From \mathcal{L}_j to \mathcal{L}_{j+1} , only one of the location counters has advanced (exception: channel rendezvous)
- ▶ \mathcal{I}_j maps each variable in M to its current value

Arithmetic and relational expressions are interpreted in states as expected; e.g. $\mathcal{L}_j, \mathcal{I}_j \models x < y$ iff $\mathcal{I}_j(x) < \mathcal{I}_j(y)$

$\mathcal{L}_j, \mathcal{I}_j \models P@1$ iff $\mathcal{L}_j(P)$ is the location labeled with 1

Evaluating other formulas $\in For_{BTL}$ in runs σ : see previous lecture

Safety Properties

Safety Properties

... are formulas of the form $[\]\phi$

- ▶ state that something good (ϕ) is **guaranteed throughout** each run
- ▶ equivalently: in $[\]\neg\psi$ something bad (ψ) **never happens**

Example

TL formula $[\](\text{critical} \leq 1)$

“it is **guaranteed throughout** each run that at most one process visits its critical section at any time”

or, equivalently:

“it will **never happen** that more than one process visits its critical section”

Applying Temporal Logic to Critical Section Problem

We want to **verify** $\square(\text{critical} \leq 1)$ as a correctness property of:

```
active proctype P() {
  do :: /* non-critical activity */
    atomic {
      !inCriticalQ;
      inCriticalP = true
    }
    critical++;
    /* critical activity */
    critical--;
    inCriticalP = false
  od
}

/* similarly for process Q */
```

Alternative 1: lt1 in model file

1. add definition of TL formula to PROMELA file

Example `lt1 s { [](critical <= 1) }`

General `lt1 <name> { <TL formula> }`

can define more than one formula

2. load PROMELA file in JSPIN
3. ensure Safety is selected
4. select Verify
 - ▶ JSPIN always selects first formula
 - ▶ use command line `./pan -N <name>` to select arbitrary formulas
5. (if necessary) select Stop to terminate too long verification

Demo: `safety1.pml`

lt1 definitions not part of Ben Ari's book ($SPIN \geq 6$): ignore 5.3.2, etc.

Model Checking a Safety Property using JSPIN

Alternative 2: edit 'LTL fomula' field of JSPIN

1. load PROMELA file in JSPIN (not necessarily containing `ltl ...`)
2. enter `[](critical <= 1)` in LTL text field of JSPIN
3. select Translate to create a 'never claim', corresponding to the negation of the formula
4. ensure Safety is selected
5. select Verify
6. (if necessary) select Stop to terminate too long verification

Demo: safety1.pml

Never Claims: Processes trying to show user wrong

Büchi automaton, as PROMELA process, for negated property

1. Negated TL formula translated to 'never' process
2. accepting locations in Büchi automaton represented with help of **accept** labels ("acceptxxx:")
3. If one of these reached infinitely often, the orig. property is violated

Example (Never claim for $\langle \rangle p$, simplified for readability)

```
never { /* !(<>p) */
  accept_xyz: /* passed  $\infty$  often iff !(<>p) holds */
  do
  :: (!p)
  od
}
```

Model Checking against Temporal Logic Property

Theory behind SPIN

1. Represent the **interleaving** of all processes as a single automaton (**only one** process advances in each step), called \mathcal{M}
2. Construct Büchi automaton (never claim) $\mathcal{N}\mathcal{C}_{\neg\phi}$ for **negation** of TL formula ϕ to be verified

3. If

$$\mathcal{L}^\omega(\mathcal{M}) \cap \mathcal{L}^\omega(\mathcal{N}\mathcal{C}_{\neg\phi}) = \emptyset$$

then ϕ holds in \mathcal{M} ,

otherwise we have a counterexample

4. To check $\mathcal{L}^\omega(\mathcal{M}) \cap \mathcal{L}^\omega(\mathcal{N}\mathcal{C}_{\neg\phi})$ construct **intersection** automaton (**both** automata advance in each step) and search for accepting run

Model Checking a Safety Property using Web Interface

1. add definition of TL formula to PROMELA file

Example `ltl s { [](critical <= 1) }`

General `ltl <name> { <TL formula> }`

can define more than one formula

2. load PROMELA file into web interface
3. ensure **Safety** is selected
4. enter name of LTL formula in according field
5. select Verify

Demo: safety1.pml

Model Checking a Safety Property using SPIN directly

Command Line Execution (Alt. 1)

Make sure `ltl <name> { <TL formula> }` is in `<file>.pml`

```
> spin -a <file>.pml
> gcc -DSAFETY -o pan pan.c
> ./pan -N <name>
```

Demo: target/safety1.pml

- ▶ The '`ltl <name> { <TL formula> }`' construct must be part of your lab submission!

Model Checking a Safety Property using SPIN directly

Command Line Execution (Alt. 2)

Write *negated* TL formula in file *<formulafile>.PRP* (first line)

```
> spin -a -F <formulafile>.PRP <file>.pml  
> gcc -DSAFETY -o pan pan.c  
> ./pan
```

- ▶ some platforms have problems with -F

Liveness Properties

Liveness Properties

... formulas of the form $\langle \rangle \phi$

- ▶ state that something good (ϕ) **eventually happens** in each run

Example

$\langle \rangle \text{csp}$

(with `csp` a variable only true in the critical section of P)

“in each run, process P visits its critical section **eventually**”

Applying Temporal Logic to Starvation Problem

We want to **verify** $\langle \text{csp} \rangle$ as a correctness property of:

```
active proctype P() {
  do :: /* non-critical activity */
    atomic {
      !inCriticalQ;
      inCriticalP = true
    }
    csp = true;
    /* critical activity */
    csp = false;
    inCriticalP = false
  od
}

/* similarly for process Q */
/* there, using csq */
```

Model Checking a Liveness Property using JSPIN

1. open PROMELA file `liveness1.pml`
2. enter `<>csp` in 'LTL fomula' field
3. select Translate to create a 'never claim', corresponding to the negation of the formula
4. ensure that **Acceptance** is selected
(SPIN will search for *accepting* cycles through the never claim)
5. *for the moment* uncheck Weak Fairness (see discussion below)
6. select Verify

Alternative to 2. and 3., write

```
ltl 1 { <>csp }
```

in PROMELA file (as first ltl formula).

Verification Fails

Demo: `start/liveness1.pml`

Verification fails!

Why?

The liveness property on one process “had no chance”.

Not even weak fairness was switched on!

Model Checking Liveness with Weak Fairness using JSPIN

Always check **Weak fairness** when verifying liveness

1. open PROMELA file
2. enter `<>csp` in LTL text field
3. select Translate to create a 'never claim', corresponding to the negation of the formula
4. ensure that **Acceptance** is selected
(SPIN will search for *accepting* cycles through the never claim)
5. ensure **Weak fairness** is checked
6. select Verify

Alternative to 2. and 3., write

```
ltl 1 { <>csp }
```

in PROMELA file (as first ltl formula).

Model Checking Liveness using Web Interface

1. add definition of TL formula to PROMELA file

Example `ltl 1 { <>csp }`

General `ltl <name> { <TL formula> }`

can define more than one formula

2. load PROMELA file into web interface
3. ensure **Acceptance** is selected
4. enter name of LTL formula in according field
5. ensure **Weak fairness** is checked
6. select Verify

Demo: liveness1.pml

Model Checking Liveness using SPIN directly

Command Line Execution (Alt. 1)

Make sure `ltl <name> { <TL formula> }` is in `<file>.pml`

```
> spin -a <file>.pml  
> gcc -o pan pan.c  
> ./pan -a -f [-N <name>]
```

-a acceptance cycles, -f weak fairness

Demo: start/liveness1.pml

Model Checking Liveness using SPIN directly

Command Line Execution (Alt. 2)

Write *negated* TL formula in file *<formulafile>.PRP* (first line)

```
> spin -a -F <formulafile>.PRP <file>.pml  
> gcc -o pan pan.c  
> ./pan -a -f [-N <name>]
```

-a acceptance cycles, *-f* weak fairness

Limitation of Weak Fairness

Verification fails again!

Why?

Weak fairness is too weak ...

Definition (Weak Fairness)

A run is called **weakly fair** iff the following holds:
each **continuously executable** statement is **executed eventually**.

Note that `!inCriticalQ` is **not** continuously executable!

Restriction to weak fairness is principal limitation of SPIN

The only way to show liveness of our example is to rewrite the model

Temporal Model Checking without Ghost Variables

We want to **verify mutual exclusion** without using ghost variables

```
bool inCriticalP = false , inCriticalQ = false;
```

```
active proctype P() {
  do :: atomic {
    !inCriticalQ;
    inCriticalP = true
  }
cs: /* critical activity */
  inCriticalP = false
od
}
```

```
/* similar for process Q with same label cs: */
```

```
ltl s { []!(P@cs && Q@cs) }
```

Demo: start/noGhost.pml

Label expressions often remove the need for ghost variables

- ▶ Specify liveness of `fair.pml` using labels
- ▶ Prove termination
- ▶ Weak fairness is needed, *and sufficient*

Demo: `target/fair.pml`

Literature for this Lecture

Ben-Ari Chapter 5

except Sections 5.3.2, 5.3.3, 5.4.2

(1t1 replaces #define and -f option of SPIN)